

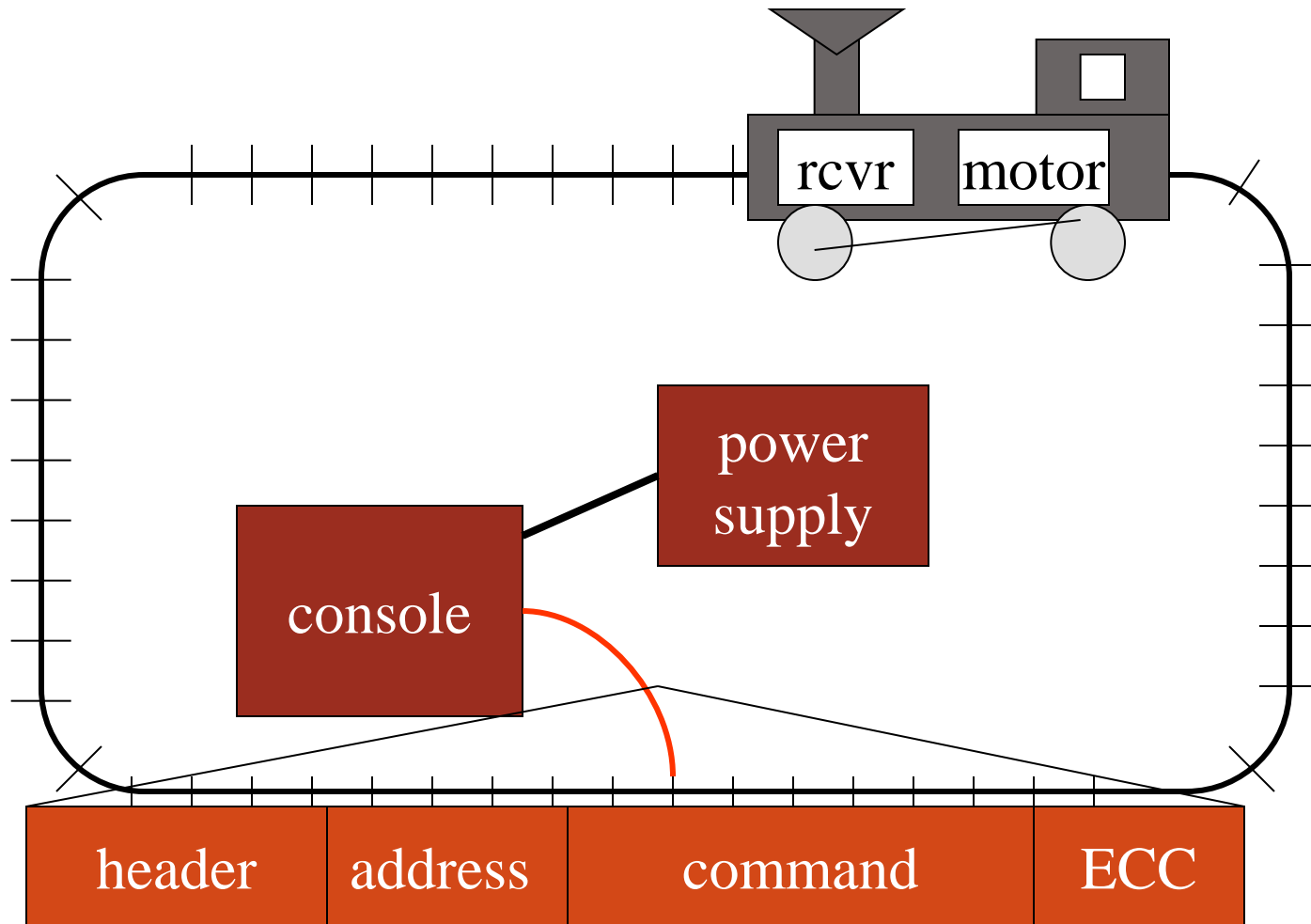
Example: Model Train Controller

Purposes of example:

Follow a design through several levels of abstraction.

Gain experience with UML.

Model train setup



Requirements

- Console can control 8 trains on 1 track.
- Throttle has at least 63 levels.
- Inertia control adjusts responsiveness with at least 8 levels.
- Emergency stop button.
- Error detection scheme on messages.

Requirements form

name	model train controller
purpose	control speed of ≤ 8 model trains
inputs	throttle, inertia, emergency stop, train #
outputs	train control signals
functions	set engine speed w. inertia; emergency stop
performance	can update train speed at least 10 times/sec
manufacturing cost	\$50
power	wall powered
physical size/weight	console comfortable for 2 hands; < 2 lbs.

Conceptual specification

- Before we create a detailed specification, we will make an initial, simplified specification.
 - Gives us practice in specification and UML.
 - Good idea in general to identify potential problems before investing too much effort in detail.

Basic system commands

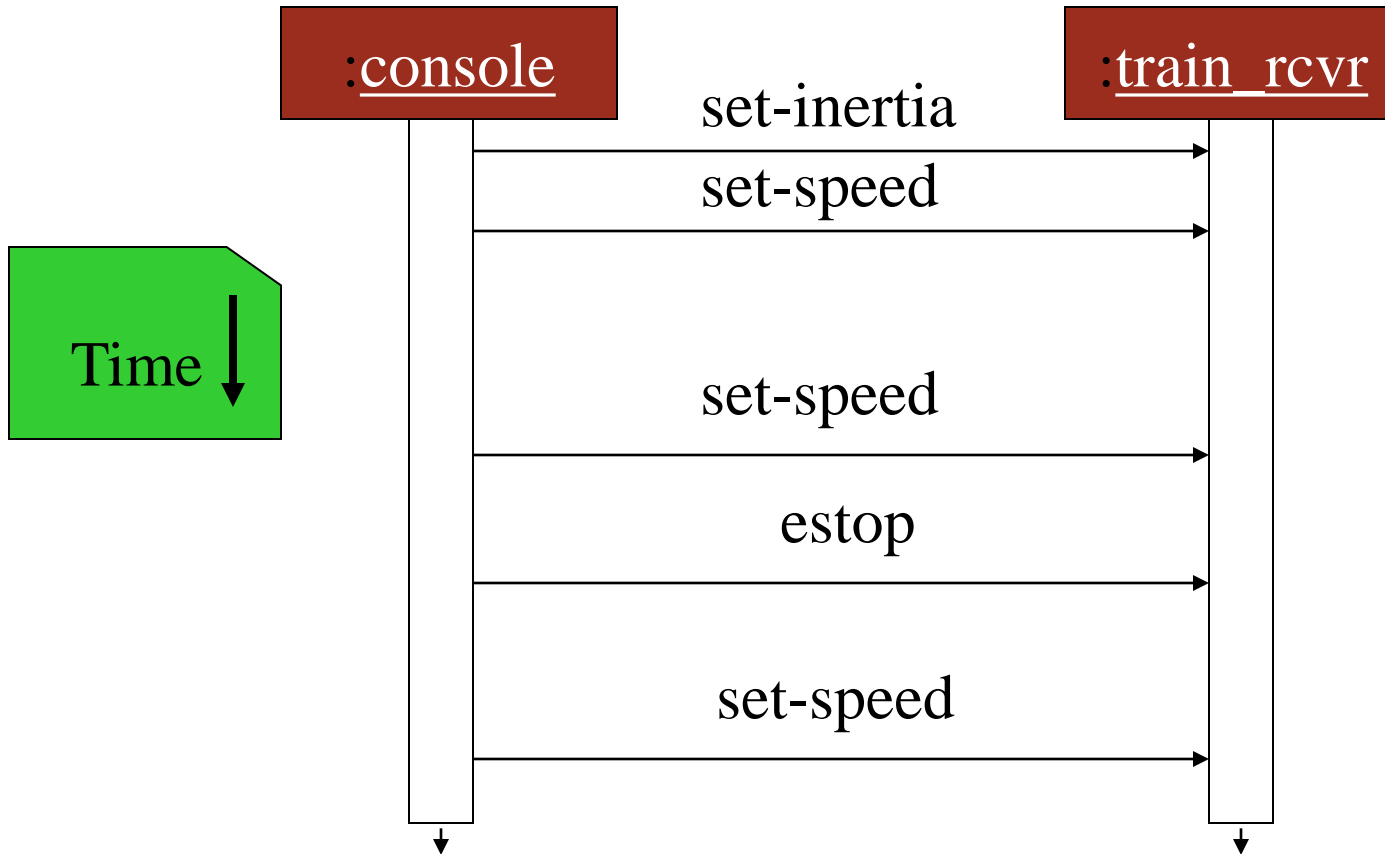
command name parameters

set-speed speed (positive/negative)

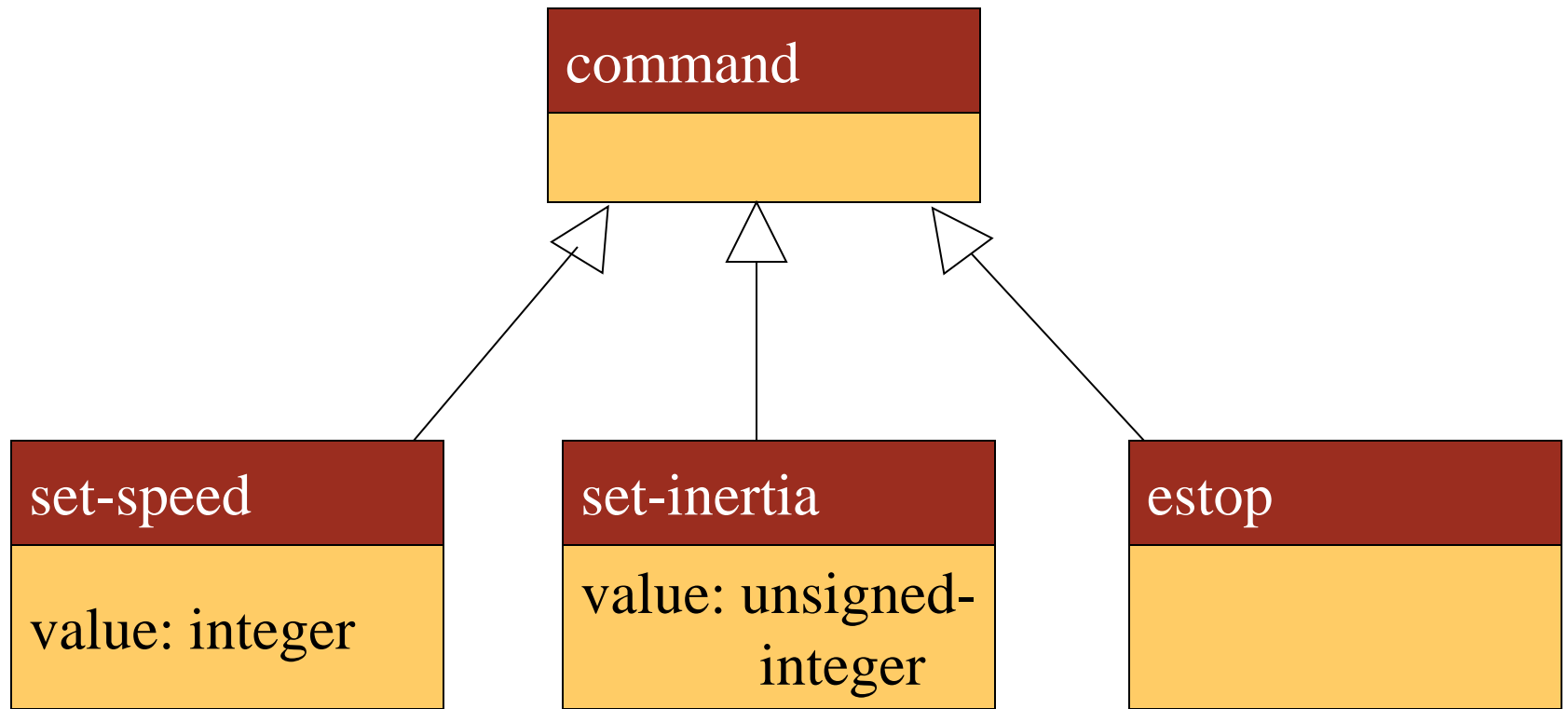
set-inertia inertia-value (non-negative)

estop none

Typical control sequence



Message classes

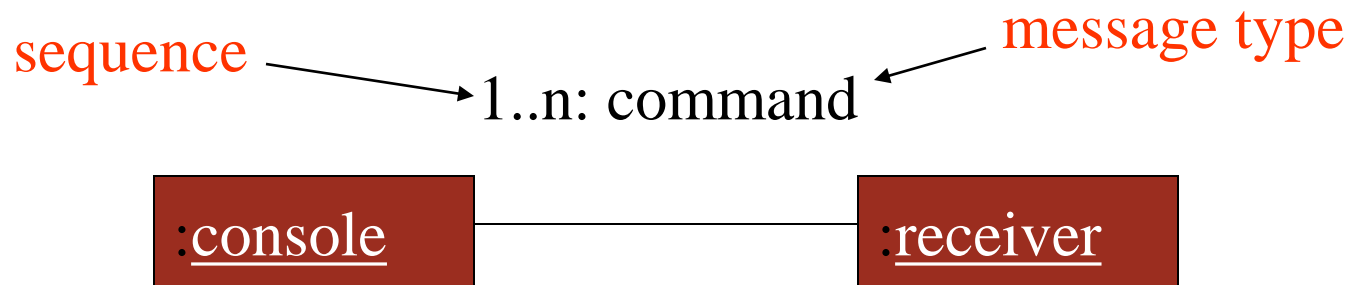


Roles of message classes

- Implemented message classes derived from message class.
 - Attributes and operations will be filled in for detailed specification.
- Implemented message classes specify message type by their class.
 - May have to add type as parameter to data structure in implementation.

Subsystem collaboration diagram

Shows relationship between console and receiver
(ignores role of track):



System structure modeling

- Some classes define non-computer components.
 - Denote by *name.
- Choose important systems at this point to show basic relationships.

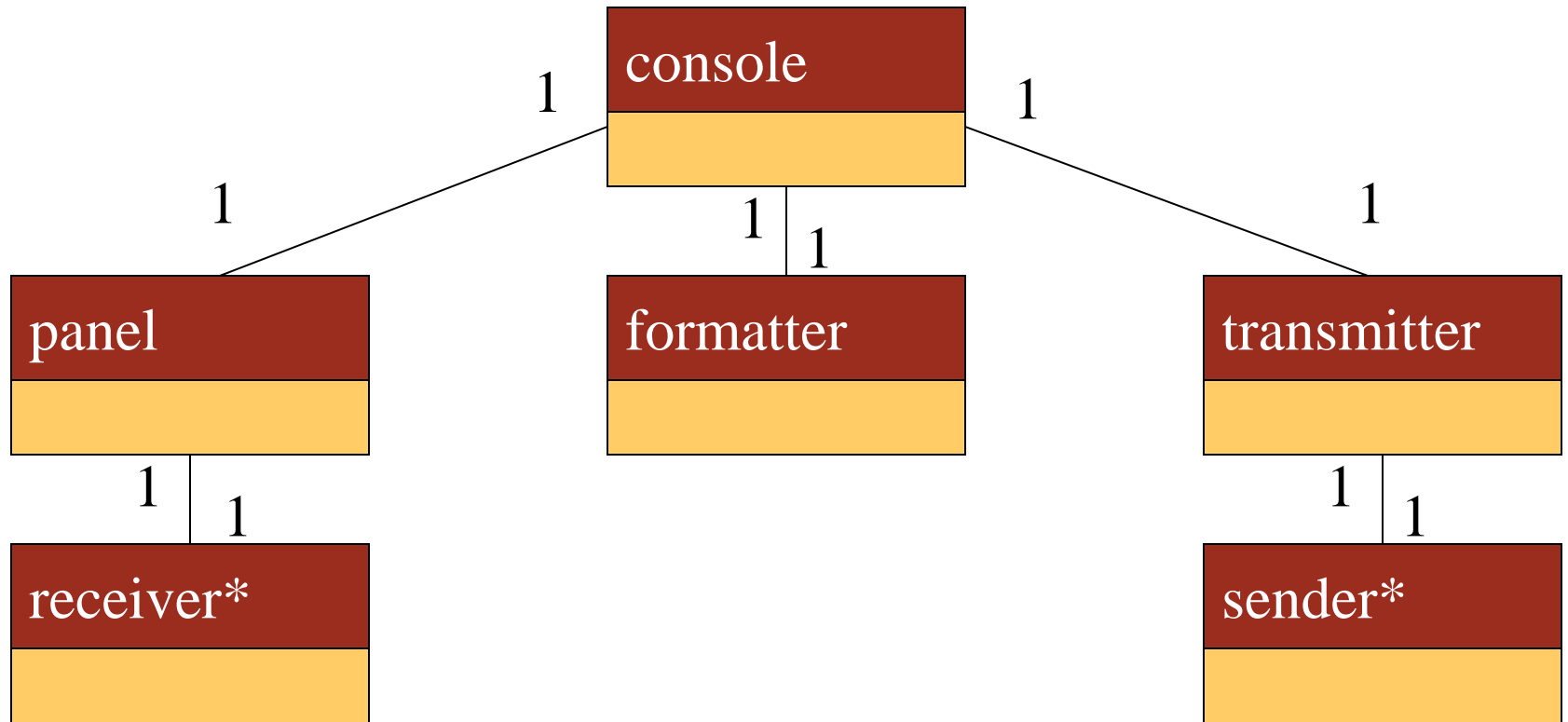
Major subsystem roles

- **Console:**
 - read state of front panel;
 - format messages;
 - transmit messages.
- **Train:**
 - receive message;
 - interpret message;
 - control the train.

Console class roles

- **panel**: describes analog knobs and interface hardware.
- **formatter**: turns knob settings into bit streams.
- **transmitter**: sends data on track.

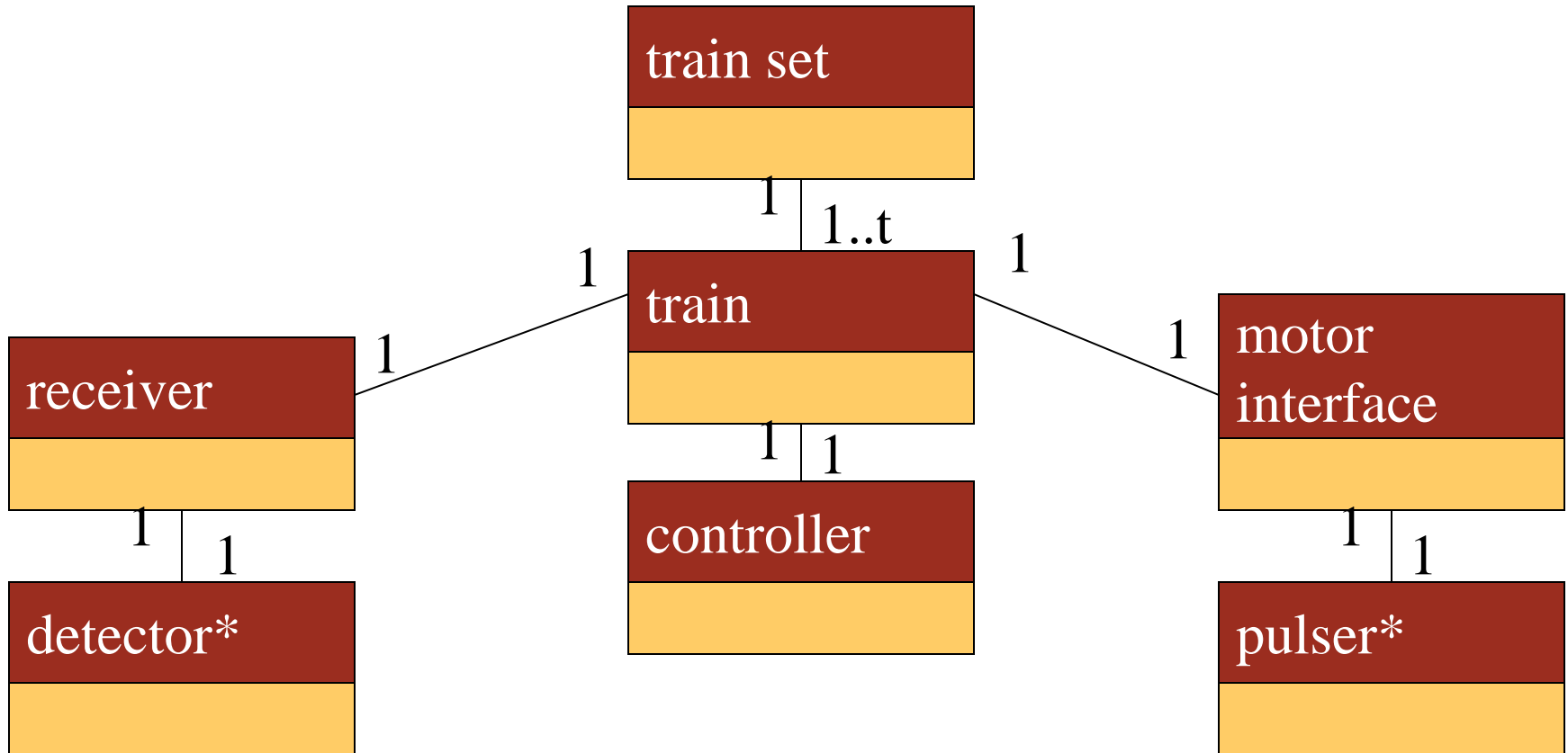
Console system classes



Train class roles

- **receiver**: digitizes signal from track.
- **controller**: interprets received commands and makes control decisions.
- **motor interface**: generates signals required by motor.

Train system classes

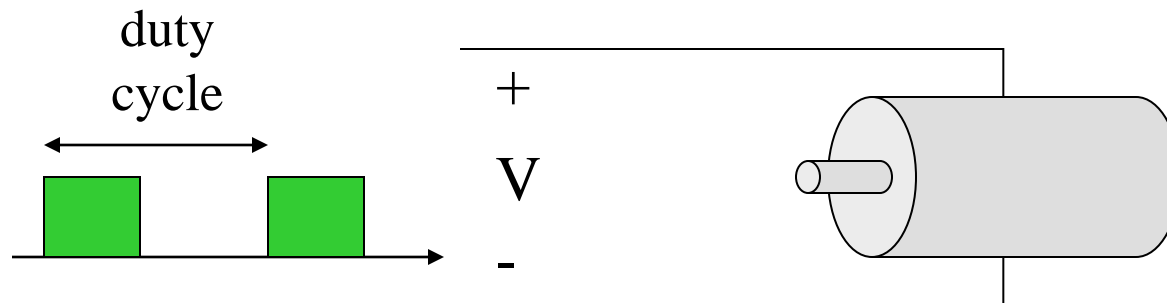


Detailed specification

- We can now fill in the details of the conceptual specification:
 - more classes;
 - behaviors.
- Sketching out the spec first helps us understand the basic relationships in the system.

Train speed control

- Motor controlled by pulse width modulation:



Train system analog physical object classes

knobs*

train-knob: integer

speed-knob: integer

inertia-knob: unsigned-integer

emergency-stop: boolean

set_knobs()

pulser*

pulse-width: unsigned-integer

direction: boolean

sender*

send-bit()

detector*

read-bit() : integer

Class descriptions

- panel class defines the controls.
 - new-settings() behavior reads the controls.
- motor-interface class defines the motor speed held as state.

Panel and motor interface classes

panel
train-number() : integer speed() : integer inertia() : integer estop() : boolean new-settings()

motor-interface
speed: integer

Class descriptions

- transmitter class has one behavior for each type of message sent.
- receiver function provides methods to:
 - detect a new message;
 - determine its type;
 - read its parameters (estop has no parameters).

Transmitter and receiver classes

transmitter

send-speed(adrs: integer,
speed: integer)
send-inertia(adrs: integer,
val: integer)
send-estop(adrs: integer)

receiver

current: command
new: boolean

read-cmd()
new-cmd() : boolean
rcv-type(msg-type:
command)
rcv-speed(val: integer)
rcv-inertia(val:integer)

Formatter class description

- Formatter class holds state for each train, setting for current train.
- The `operate()` operation performs the basic formatting task.

Control input cases

- Use a soft panel to show current panel settings for each train.
- Changing train number:
 - must change soft panel settings to reflect current train's speed, etc.
- Controlling throttle/inertia/estop:
 - read panel, check for changes, perform command.

Formatter class

formatter

current-train: integer

current-speed[ntrains]: integer

current-inertia[ntrains]:

 unsigned-integer

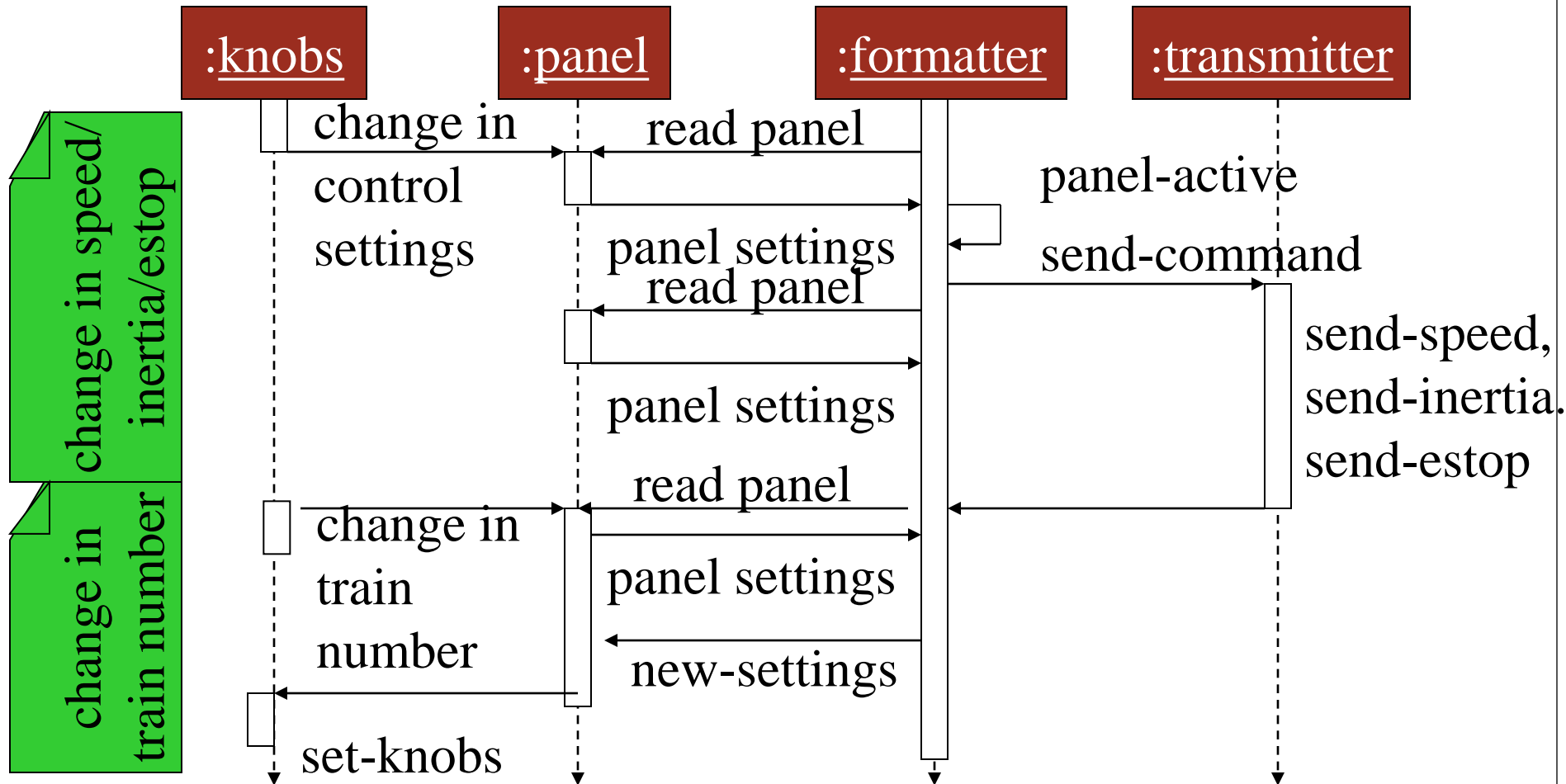
current-estop[ntrains]: boolean

send-command()

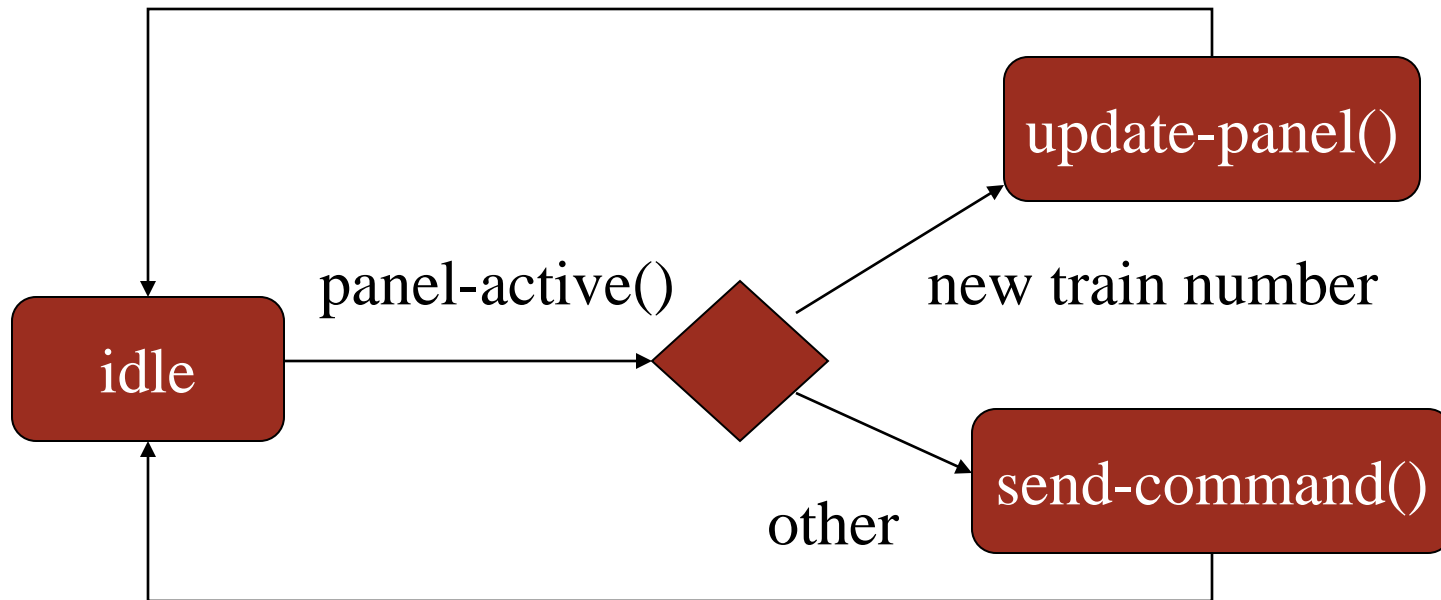
panel-active() : boolean

operate()

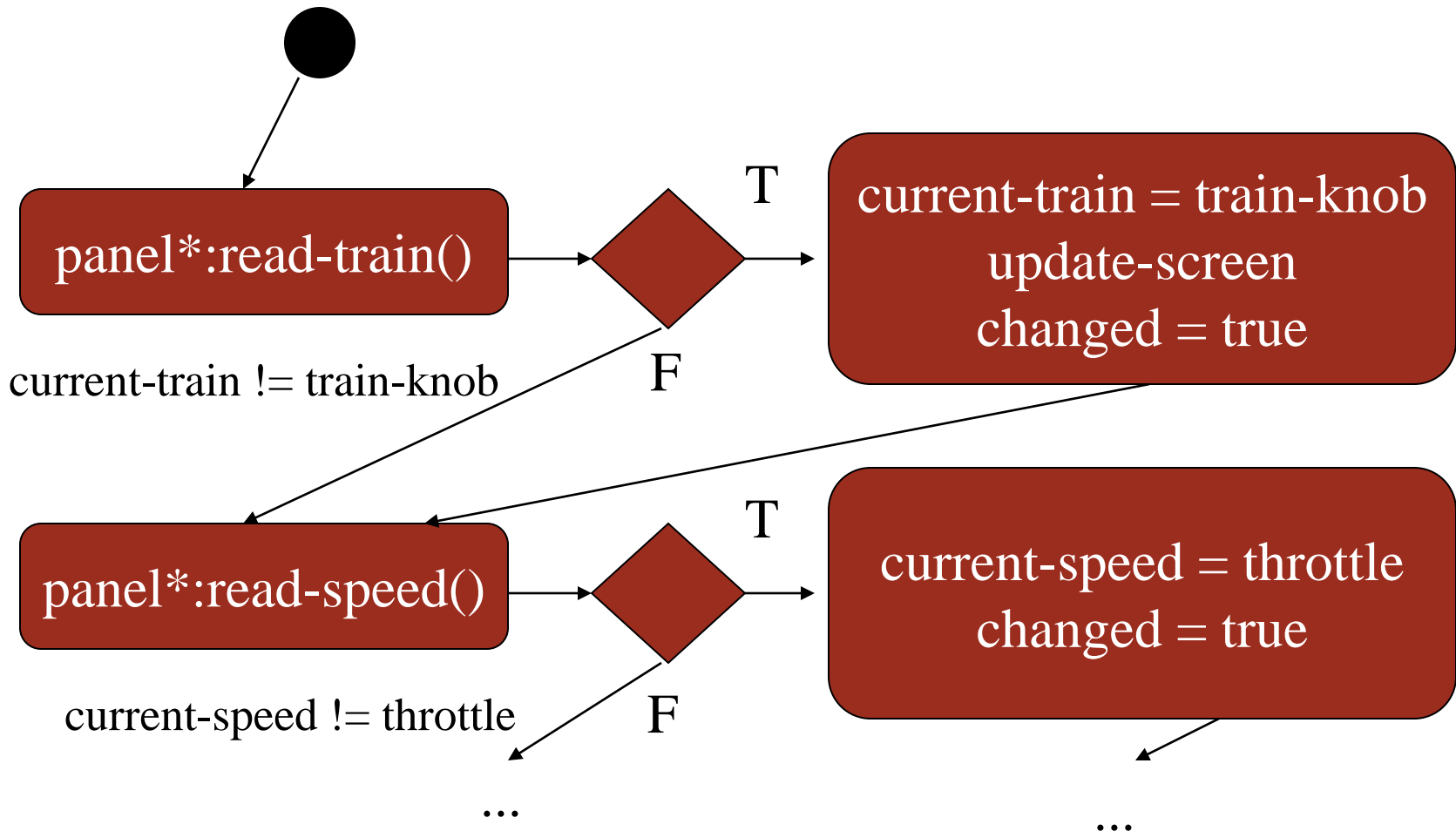
Control input sequence diagram



Formatter operate behavior (in the formatter class)



Panel-active behavior (in the formatter class)



Train controller class

controller

current-train: integer

current-speed[ntrains]: integer

current-direction[ntrains]: boolean

current-inertia[ntrains]:
unsigned-integer

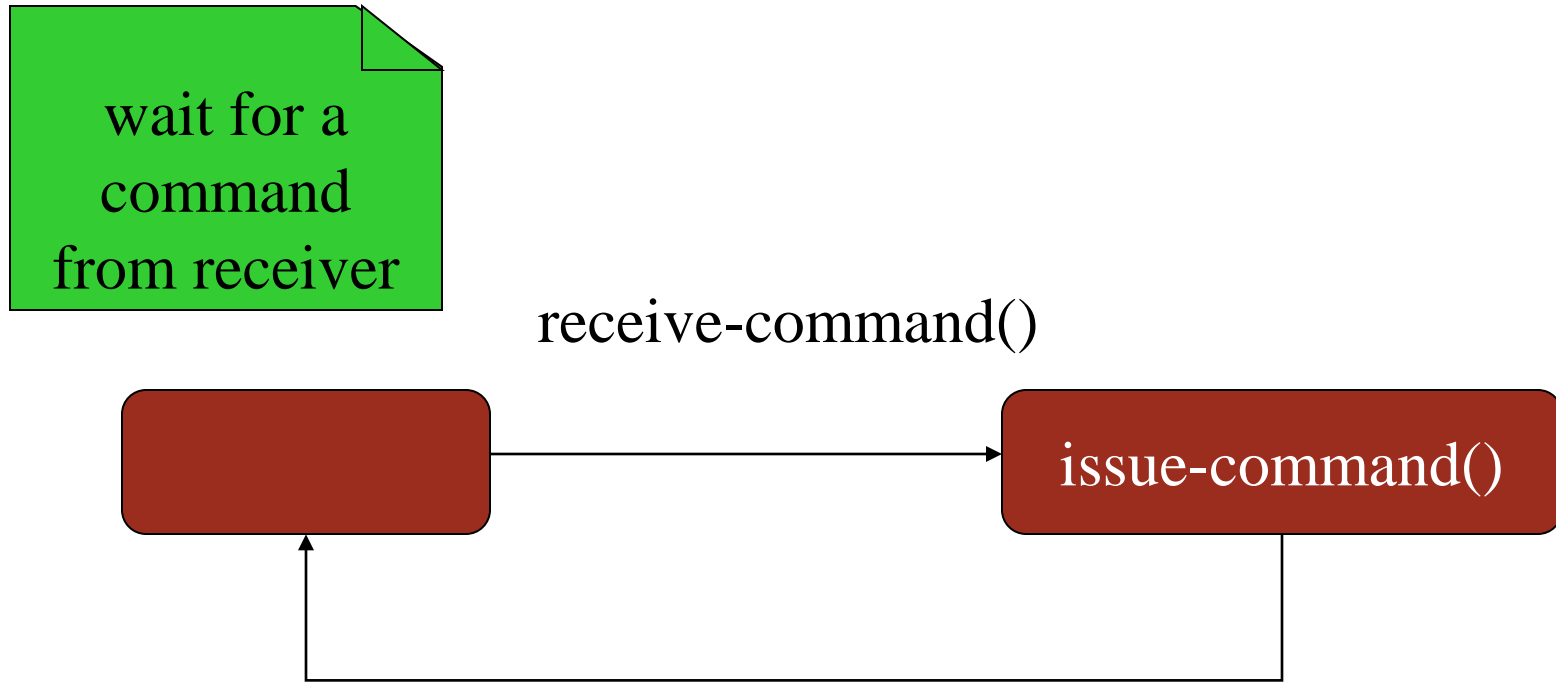
operate()

issue-command()

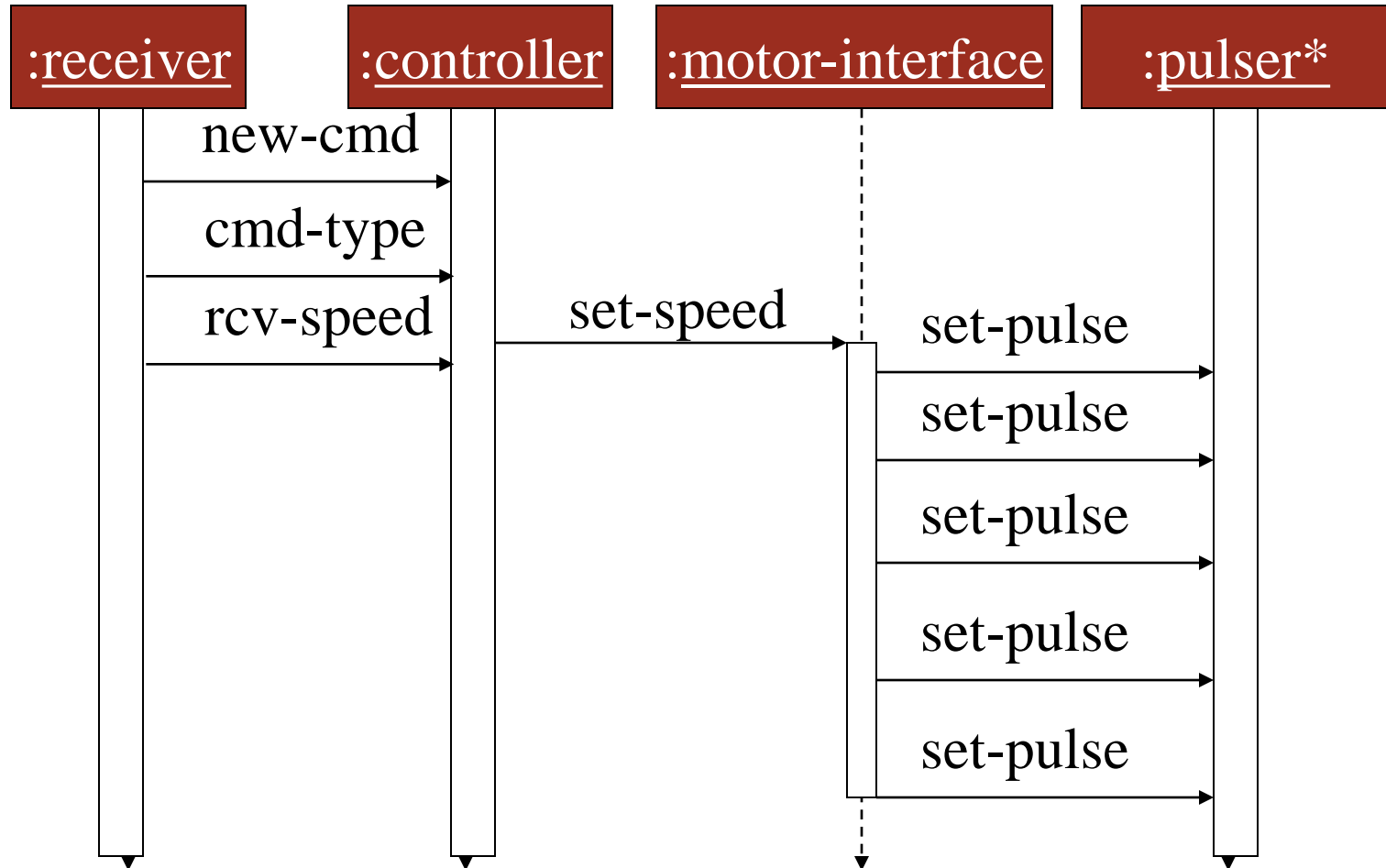
Setting the speed

- Don't want to change speed instantaneously.
- Controller should change speed gradually by sending several commands.

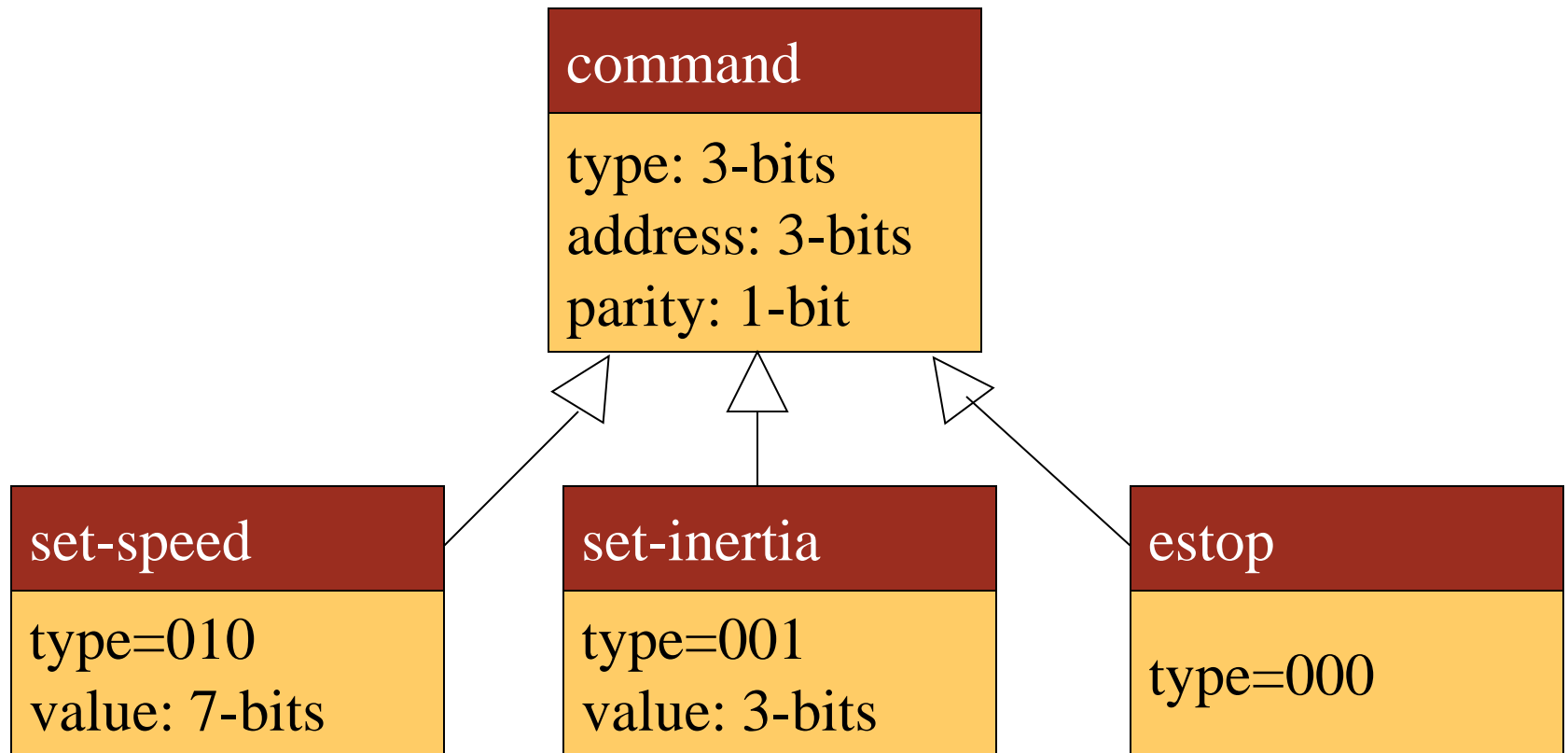
Controller operate behavior



Sequence diagram for set-speed command



Refined command classes



Summary

- Separate specification and programming.
 - Small mistakes are easier to fix in the spec.
 - Big mistakes in programming cost a lot of time.
- You can't completely separate specification and architecture.
 - Make a few tasteful assumptions.